ＡＤＳＰ３２４－２９

ＰＩＤ制御ライブラリ

ユーザーズ・マニュアル

中部電機株式会社

目　　次

１.概要 1

２.納入ディスク内容 1

３.ライブラリのインストール 2

４.関数一覧 2

５.関数リファレンス 3

６.テストプログラムジェネレータ 8
　６.１　ソースの作成 8
　６.２　プログラムのコンパイル・リンク 9
　６.３　プログラムの実行 10
　６.４　プログラムの動作モニタ 11
　６.５　ハードウェアの変更が有る場合 11

付録　　Ａ：ＰＩＤ演算式の展開 12
付録　　Ｂ：管理構造体の作業領域の詳細 14
付録　　Ｃ：コンパイラのバージョンアップ 16
付録　　Ｄ：互換Ｃ関数ソース 17

１．概要

１入力１出力の制御則の代表である ＰＩＤ制御は、理論と実績に裏付けられた上、ロバスト性、安定性、少ない演算量等、さまざまな特色を持っており、現在でも広く実用とされている制御の代表的な存在となっています。

本ライブラリは、ＡＤＳＰ３２ｘシリーズ、浮動小数点演算ＤＳＰ上でＰＩＤ制御を実現するライブラリです。 本ライブラリを用いる事で、ＰＩＤ制御ユニットが簡単に実現出来る上、これの応用により、カスケード制御等も実現できます。

また、テストプログラム自動生成ツールが 付属していて、ゲイン、制御周期等、必要最小限のパラメータを入力するだけで、即、実行可能なプログラムソースが自動生成されます。これを使用してユーザアプリケーションのテンプレートとしても利用できます。

２．納入ディスク内容

○媒体	ＡＤＳＰ３２２－７９	3.5 インチＦＤ（1.2MB）
	ＡＤＳＰ３２４－２９	3.5 インチＦＤ（1.44MB）

○内容	PIDLIB.LIB	ＰＩＤ演算ライブラリ
	PIDLIB.SRC	ライブラリソースファイル
	PID.H	Ｃ言語用ヘッダファイル
	DEMOGEN.EXE	テストプログラム自動生成ツール
	PIDMON.C	テストプログラムモニター（ソースファイル）
	PIDMON.EXE	テストプログラムモニター（実行ファイル）
	PIDMON.PRM	テストプログラムパラメータファイル
	INSTALL.BAT	インストーラバッチコマンド

３．ライブラリのインストール

本ライブラリのインストールは以下のファイルを、Ｃコンパイラの標準ライブラリ(RTS?.LIB)、及びヘッダファイル(STDLIB.H 等)の格納されているディレクトリ（即ち、環境変数 C_DIR が指標しているディレクトリ）へコピーします。

・PIDLIB.LIB
・PIDLIB.SRC
・PID.H

ＤＳＰ用コンパイラを正しくインストールしてから、本製品のインストール用バッチコマンド、INSTALL.BAT を起動してください。

＜参考＞
PIDLIB.SRC は直接には利用されませんが、コンパイラの版が変わった場合等に利用する場合があります。付録Ｃを参照してください。
他のファイル（DEMOGEN.EXE、PIDMON.*）は、テストプログラムの項を参照してください。

４．関数一覧

本ライブラリセットには以下の関数が含まれています。

PIDinit()	ＰＩＤ制御管理構造体を初期化します。
PIDgain()	ＰＩＤ制御のゲインを変更します。
PIDcalc()	指令値、ＦＢ値から、ＰＩＤ制御計算をします。
PIDcalc2()	ＦＢ誤差値から、ＰＩＤ制御計算をします。

５．関数リファレンス

==========
○管理構造体 PID_CTRL
==========

本ライブラリの関数は複数のＰＩＤ制御ユニットを構築できるよう、各ＰＩＤ制御ユニットは構造体で管理しています。この構造体はヘッダファイル PID.H に定義されていて、typedef 宣言子で PID_CTRL と宣言されています。この構造体は以下に述べるメンバーを持っています。

float ref; 指令値に相当する実数データを入力します。

float fb; フィードバック値に相当する実数値を入力します。

float err; 誤差値に相当する実数値を入力または出力します。PIDcalc()では出力となり、誤差値として err = ref - fb が計算され、ここに格納されます。PIDcalc2()では入力となりユーザが ref - fb を計算し入力します。

float cv; 制御値が出力されます。PIDcalc(),PIDcalc2()の関数戻り値と同じです。再参照する場合に利用します。

float Pgain; 比例項ゲイン（＊１）を設定します。

float Igain; 積分項ゲイン（＊１）を設定します。

float Dgain; 微分項ゲイン（＊１）を設定します。

（＊１）各ゲインは連続時間系でのゲイン値を設定してください。詳しくは付録を参照してください。

float tds; 制御周期を指定します。単位は秒です。この変数はＰＩＤ計算式のＺ変換計算に使用されます。ハード的な制御周期自体を決定するものではありませんから注意が必要です。タイマ／外部割り込み等で決定される実際の制御周期との相違があると理論値通りの制御ゲインが得られません。

float work[22];ＰＩＤ計算ルーチンが利用する作業領域です。ユーザはアクセスしないでください。

==========

○PIDinit()　　ＰＩＤ制御管理構造体の初期化

==========

関数定義void　　PIDinit(PID_CTRL *pid);

機能　　ＰＩＤ管理構造体の初期化をします。
PIDcalc()、PIDcalc2()にて制御計算を行う前には、必ずこの初期化を行って下さい。
初期化をせずに PIDcalc()等を呼び出すと最悪の場合ＤＳＰの暴走等を招きます。

入力引数pid->Pgain：比例項ゲイン
pid->Igain：積分項ゲイン
pid->Dgain：微分項ゲイン
pid->tds　：計算周期。単位［秒］

出力　　特にありません。ＰＩＤ管理構造体内の作業領域は全て初期化され、計算に備えます。

使用例

```
#include <pid.h>
PID_CTRL        pid;

main()
{
        pid.Pgain = 5;          /* P gain set */
        pid.Igain = 100;        /* I gain set */
        pid.Dgain = 0.0001;     /* D gain set */
        pid.tds = 0.001;        /* time constant (1mSec) set */
        PIDinit(&pid);
}
```

==========

○PIDgain()　　ＰＩＤ制御のゲインの変更

==========

関数定義void　　PIDgain(PID_CTRL *pid);

機能　　ＰＩＤ管理構造体のゲインの変更をします。
ＰＩＤゲインを若干変更する場合に利用でき、積分項をクリアせずにゲインが変更できます。但し、微分項がある程度以上のゲインの場合や、比例・積分項のゲイン変更量が著しい場合は、乱調を来す場合がありますから、PIDinit()を用いて下さい。

入力引数pid->Pgain：比例項ゲイン
pid->Igain：積分項ゲイン
pid->Dgain：微分項ゲイン
pid->tds　：計算周期。単位［秒］

出力　　特にありません。ＰＩＤ管理構造体内のゲイン定数が変更されます。

使用例

```
#include <pid.h>
PID_CTRL        pid;

main()
{
        pid.Pgain = 5;              /* P gain set */
        pid.Igain = 100;            /* I gain set */
        pid.Dgain = 0;              /* D gain set */
        pid.tds = 0.001;            /* time constant (1mSec) set */
        PIDinit(&pid);

        /* ・・・制御計算実行 */

        pid.Pgain = 4.9;            /* P gain change */
        pid.Igain = 103;            /* I gain change */
        PIDgain(&pid);

        /* ・・・制御計算続行 */
}
```

==========

○PIDcalc()　　ＰＩＤ制御計算（１）

==========

関数定義float　PIDcalc(PID_CTRL *pid);

機能　ＰＩＤ制御演算を行います。演算データとして、指令値と、フィードバック値が別々に測定できる場合に利用できます。

入力引数pid->ref 指令値を入力します。
　　pid->fb　　フィードバック値を入力します。

出力　　関数戻り値　　制御量が出力されます。
　　pid->err 誤差値 pid->ref - pid->fb が出力されます。
　　pid->cv　　制御量が出力されます。関数戻り値と同じです。
　　制御量を再参照する場合に利用できます。

使用例

```
#include"pid.h"

PID_CTRL        pidctrl = {0,0,0,0,1,2,0.005,0.001};
extern void     TimeSync(void);         /* sync to time period */
extern float    Reference(void);        /* get reference value */
extern float    Feedback(void);         /* get feedback value */
extern void     CtrlOutput(float);      /* output controll value */

main()
{
        PIDinit(&pidctrl);
        for(;;){
                TimeSync();
                pidctrl.ref = Reference();
                pidctrl.fb = Feedback();
                CtrlOutput(PIDcalc(&pidctrl));
        }
}
```

==========

○PIDcalc2()　　ＰＩＤ制御計算（２）

==========

関数定義float　　PIDcalc2(PID_CTRL *pid);

機能　　ＰＩＤ制御演算を行います。演算データとして、誤差値が直接測定できる場合に利用できます。

入力引数pid->err 誤差値を入力します。

誤差値　＝　指令値　－　フィードバック値

出力　　関数戻り値　　制御量が出力されます。

pid->cv　　制御量が出力されます。関数戻り値と同じです。
制御量を再参照する場合に利用できます。

使用例

```
#include "pid.h"

PID_CTRL        pidctrl = {0,0,0,0,1,2,0.005,0.001};
extern void     TimeSync(void);          /* sync to time period */
extern float    Errorval(void);          /* get error value */
extern void     CtrlOutput(float);       /* output controll value */

main()
{
        PIDinit(&pidctrl);
        for(;;){
                TimeSync();
                pidctrl.err = Errorval();
                CtrlOutput(PIDcalc(&pidctrl));
        }
}
```

６．テストプログラムジェネレータ

本ライブラリを用いて標準的なＰＩＤ制御を行うサンプルプログラムのソースファイルを自動生成するツールです。次の手順で作業を行ってください。尚、ここでは、説明の簡略化の為以下の環境までセットアップされているものとして説明を進めます。

１．Cコンパイラは既にインストールされ利用可能な状態にセットアップされている事とします。
２．ＤＳＰの付属ソフトウェアのライブラリ CBIOS.LIB (CBIOS322.LIB / CBIOS324.LIB) がリンカーから参照可能である事とします。
３．ＤＳＰの付属ソフトウェアのリンカーレスポンスファイル USERB.CMD 又は、ＭＳＣアダプタ（Cアダプタ）の同 EXE.CMD がカレントディレクトリに格納されている事とします。
４．ＤＳＰの付属ソフトウェアの CBIOS.H(CBIOS322.H/CBIOS324.H) がコンパイラから参照可能である事とします。

又説明の便宜上、カレントドライブはＡ：である事とします。

６.１　ソースの作成

ＤＯＳのプロンプトから DEMOGEN.EXE を起動してください。

```
A>demogen ・
```

画面に表示されるガイダンスに従って次のパラメータを入力します。

・ＤＳＰボードタイプ (ADSP322-00 / ADSP322-50 / ADSP324-00)
・ＰＩＤ制御ユニット数
・制御周期
・各ユニットの
　　指令値入力Ａ／Ｄ　CH 番号
　　ＦＢ値入力Ａ／Ｄ　CH 番号
　　制御出力Ｄ／Ａ CH 番号
　　Ｐ・Ｉ・Ｄ各項のゲイン
・使用するＡ／Ｄ各 CH のレンジ (±10V、±5V)

・使用するＤ／Ａ各 CH のレンジ (±10V、±5V、±2.5V 等)
・ＭＳＣアダプタの有無

以上の入力が終るとカレントディレクトリに以下のファイルが作成されます。

SMPL.C	ソースファイルです。
SMPL.LNK	リンカーに渡すレスポンスファイルです。
SMPL.MK	プログラムメンテナンスファイルです。
SMPL.BAT	プログラムの実行開始するバッチコマンドです。

６.２　プログラムのコンパイル・リンク

ＭＳＣ等のプログラムメンテナー、NMAKE をお持ちの場合

A> NMAKE　-f　smpl.mk　<CR>

としてください。
それ以外の場合には以下の手順でコンパイル・リンクを行ないます。

１．コンパイラを起動しコンパイルします。

```
A>cl30　-mb　-q　smpl.c <CR>
[smpl.c]
```

コンパイラからのエラーメッセージが無ければコンパイル完了です。

２．オブジェクトのリンクをします。

付属ソフトウェアだけで実行する場合は、ＤＳＰボードの付属ディスクから USERB.CMD をカレントディレクトリにコピーして下さい。又、ＭＳＣアダプタ（Ｃアダプタ）をお使いの場合は、EXE.CMD をカレントディレクトリにコピーしてください。

次に、リンカーを起動します。

```
A>lnk30　smpl.lnk <CR>
```

リンクが完了し、カレントディレクトリに SMPL.OUT と SMPL.MAP が作成され

ています。

３．ブートモジュールの作成
ＤＳＰ付属ソウトウェアだけで実行する場合の為に、次の操作でブートモジュールを作成します。

A>stgen322　smpl　smplb <CR>

カレントディレクトリに、 SMPLB.OUT が作成されます。これは同付属ソフトウェアのひとつ、プログラムローダ LOAD322 によりプログラムを実行する場合に必要となるファイルです。（PC/AT の場合はそれぞれ、stgen324 , load324）

６．３　プログラムの実行

起動用バッチコマンド SMPL.BAT が作成されていますから、これで起動します。

A>smpl　<CR>

コマンド内では次の処理を行なっています。

reset322　/h
load322　smpl　smplb　/g

但し、PC/AT の場合はそれぞれ、reset324 , load324 となります。
RESET322 のスイッチ /h はリセット後ＤＳＰが勝手に走らないようホールドします。こうしないと電源投入後ＤＳＰにまだ一度もプログラムをロードしていない場合、ＤＳＰのＲＡＭの内容は不定ですから、ＤＳＰは暴走状態になってしまいます。
LOAD322 でプログラム本体と、ブートモジュールをロードします。スイッチ /g は、ロード後直ちにＤＳＰに実行を開始させるものです。

これで、ＤＳＰは実行を開始し、直ちに、ＰＩＤ制御を開始します。

６.４　プログラムの動作モニタ

ＤＳＰの制御周期が充分に長い場合（例えば 0.1 秒等）では、モニタプログラムの実行により、制御の様子を画面に数値で表示する事ができます。PIDMON.EXE はその為に作成されたプログラムです。

SMPL.BAT では次のように、PIDMON.EXE を起動しています。

```
A>pidmon <CR>
```

モニタの終了は ESC キーを入力します。モニタ表示は終了しますが、制御は継続しています。

ＤＳＰを停止するには、

```
A>reset322　/h
```

とします。（PC/AT の場合は reset324）

６.５　ハードウェアの変更が有る場合

ホストのメモリ環境の関係で、ＤＳＰボードのメモリーをホストに割り付ける際のアドレスを変更している場合、モニタプログラム PIDMON の動作をそれに併せる必要があります。PIDMON.EXE は実行時にカレントディレクトリの PIDMON.PRM を参照し、その内容に従って DSP をアクセスします。例えばＤＳＰをホストのメモリー番地 Ｃ００００Ｈ番地に割り付けた場合、PIDMON.PRM の内容は、

```
c000
```

とします。ＤＳＰの設定を変更した場合はそれに併せてこのファイルを変更してください。

ＤＳＰボードの出荷時設定は以下の通りとなっています。

ADSP322-00	C000
ADSP322-50	C000
ADSP324-00	E000

付録　Ａ：ＰＩＤ演算式の展開

本ライブラリは、連続時間系のＰＩＤ演算式を元に、双一次変換によりＺ変換した離散化ＰＩＤ演算により制御計算を行います。以下に計算式を展開します。

連続時間系での制御誤差量、及びＰＩＤ演算式は、

$$E(t) = R(t) - F(t) \qquad (1)$$

ここで　E(t)：時刻 t に於ける誤差(Error)
R(t)：時刻 t に於ける指令値(Reference)
F(t)：時刻 t に於けるフィードバック値(Feedback)

$$C(t) = KP \cdot E(t) + KI \cdot \int E(t)dt + KD \cdot \frac{dE(t)}{dt} \qquad (2)$$

C(t)：時刻 t に於ける制御量(Controll)
KP　：比例項ゲイン
KI　：積分項ゲイン
KD　：微分項ゲイン

で表されます。
式（２）をラプラス変換して、

$$C(s) = KP \cdot E(s) + KI \cdot \frac{E(s)}{s} + KD \cdot E(s) \cdot s \qquad (3)$$

s：ラプラス演算子

とします。式（３）を、双一次変換により、連続系から離散系へＺ変換します。双一次変換は、

$$s = \frac{2}{T} \cdot \frac{1 - Z^{-1}}{1 + Z^{-1}} \qquad (4)$$

T ：サンプリング（制御）周期
Z ：Z演算子

ですから、これを式（3）に代入すると、

$$C(Z) = KP \cdot E(Z) + KI \cdot E(Z) \cdot \frac{T}{2} \cdot \frac{1+Z^{-1}}{1-Z^{-1}} + KD \cdot E(Z) \cdot \frac{2}{T} \cdot \frac{1-Z^{-1}}{1+Z^{-1}} \qquad (5)$$

式（5）を整理して、

$$C(Z) = C(Z) \cdot Z_{-2} + K1 \cdot E(Z) + K2 \cdot E(Z) \cdot Z_{-1} + K3 \cdot E(Z) \cdot Z_{-2} \qquad (6)$$

但し、

$$K1 = PK + \frac{T}{2} \cdot KI + \frac{2}{T} \cdot KD$$

$$K2 = T \cdot KI - \frac{4}{T} \cdot KD$$

$$K3 = -KP + \frac{T}{2} \cdot KI + \frac{2}{T} \cdot KD$$

を得る事ができます。Z演算子を遅延オペレーターと解釈すると、

$$Z_{-n} \cdot f(t) = f(t-n) \qquad (7)$$

ですから、式（6）は

$$C(t) = C(t-2) + K1 \cdot E(t) + K2 \cdot E(t-1) + K3 \cdot E(t-2) \qquad (8)$$

但し、K1,K2,K3 は式（6）と同じ

と書き換える事ができます。
本ライブラリは、式（8）を用いて計算しています。

付録　Ｂ：管理構造体の作業領域の詳細

本ライブラリをＰＩＤ演算ユニットとして、ブラックボックス的に利用する場合、管理構造体の作業領域 float work[] につていてユーザは全く知る必要はありません。
しかし、この作業領域はＰＩＤ計算には必要不可欠で、かつ、計算の殆どはこの領域を使って行われています。
本付録は、本ライブラリのＰＩＤ計算処理の、具体的計算実現方法等について興味をお持ちのユーザに、具体的詳細情報を提供する為のものです。

管理構造体の、より詳細な定義は、PIDINIT.C に struct PIDcontroll_sp として定義されています。
この構造体の前半は　PID_CTRL と全く同じで、ユーザとのインタフェースを取っています。
後半は以下のメンバーになっています。

```
float     k1;           /* ＰＩＤ計算係数Ｋ１ */
float     k2;           /* ＰＩＤ計算係数Ｋ２ */
float     k3;           /* ＰＩＤ計算係数Ｋ３ */
float     *kp;          /* Ｋ３アドレス（Ｋ１～Ｋ３は連続） */
float     *errp;        /* 誤差値リングバッファポインタ */
float     *cvp;         /* 制御値リングバッファポインタ */
float     errbuf[BL];   /* 誤差値リングバッファ */
float     cvbuf[BL];    /* 制御値リングバッファ */
```

float　k1,k2,k3 は、式（８）の K1,K2,K3 に相当する変数で、PIDinit() 又は PIDgain() により Pgain , Igain , Dgain , tds から、式（６）の但し書きにより計算します。
この計算は各ゲインと時定数が決まれば一義的に決定されますから、初期化関数で一度だけ計算し、制御周期毎の計算には k1 , k2 , k3 を使用します。

float *kp; は、float k3 のアドレスを指標しており、PIDcalc()等で計算する際に k1～k3 の連続するアドレスを間接参照で利用します。つまり k3 のアドレス計算を初期化の際に１度だけ行い、それを覚えておきます。計算時はそのアドレスから、ＤＳＰの間接参照機構のデクリメント機構を使って順次参照する事でアドレス計算を省略しています。

float *errp; は誤差値リングバッファ float errbuf[] の、あるアドレスを指標しています。

誤差リングバッファは、ＤＳＰの持つサーキュラアドレッシング機構を用いてリングバッファ（サーキュラバッファと同意語）を構築しています。
バッファの長さは３ワードで、式（８）の E(t),E(t-1),E(t-2)を保存しています。
次の計算時には時刻 t が経過しているので、E(t-1)→E(t-2)、E(t)→E(t-1)となり、前回の E(t-2)の領域が不要となり、ここに今回の E(t)を格納します。
計算開始時点の float *errp; はこの、最新の E(t)のアドレスを指標しています。

float *cvp; は制御値リングバッファ float cvbuf[] の、あるアドレスを指標しています。
制御値リングバッファは、ＤＳＰの持つサーキュラアドレッシング機構を用いてリングバッファを構築しています。バッファの長さは２ワードで、式（８）の C(t-2) 及び、C(t-1) を保存しています。
計算に使用するのは、C(t-2)ですから、これを参照して C(t) を計算した後、C(t-2)の領域に C(t)を保存して計算を終了します。
次の計算時には時刻 t が経過しているので、C(t-1)→C(t-2)、C(t)→C(t-1)となります。
計算開始時点の float *cvp; はこの、C(t-2) を指標しています。

errbuf[] , cvbuf[] の、各リングバッファは実際に使用しているのはそれぞれ、３と２ワードです。しかしながら、構造体宣言では８ワードが割り付けてあります。これは以下の理由によります。
ＤＳＰの持つサーキュラアドレッシング機構によるサーキュラアドレッシングには配置の制約があり、その先頭アドレスは、サーキュラバッファの長さを決定する BK レジスタの値により定まる、あるワード境界に配置されなければなりません。ところが、Ｃ言語では（一部のコンパイラを除き）、構造体等の配置をユーザが任意に定める事が出来ません。
そこで、やむを得ず、どんな番地に配置されても、必要なワード境界とそれに続く必要なワード数を包含する、必要最小限かつ充分な領域を割付け、先の制約に合った領域のみを使用する事としています。
リングバッファ長、３の場合、その先頭アドレスは、４の倍数境界になければいけません。そこでバッファ領域としては８ワード用意し、その中に包含される４の倍数アドレスから始まる連続する３ワードを、実際に利用します。
　このアドレス計算は、PIDINIT.C の中の AddressForm() で行っています。
尚、errbuf[] と cvbuf[] の両方あわせて合計１２ワード以内にする事も出来ますが、本ライブラリでは、プログラムの可視性や、トータルのメモリ効率（データメモリは４ワード節約になるが逆にプログラムが大きくなる。）を考慮して、別々の各８ワード領域を割り付けました。

付録　Ｃ：コンパイラのバージョンアップ

本ライブラリは、Ｃ言語とアセンブラ言語で記述されています。いずれも、Ｃコンパイラ（アセンブラ）　リリース　R4.50　でコンパイル（アセンブル）されています。
将来、コンパイラがバージョンアップされた場合、又は、R4.50 以前のコンパイラで利用する場合、おそらく、基本的には関数呼び出しシーケンス等は変更されませんから、アセンブラで記述された関数（PIDcalc()等）は問題ありません。しかし、Ｃで記述された関数（PIDinit()等）は、場合によってはコンパイラメーカが供給する基本演算ルーチンが変更される事により、未定義になる場合があります。
その場合は、本ライブラリのソースファイル PIDLIB.SRC を元に使用しているコンパイラー（アセンブラ）で再コンパイル（アセンブル）し、ライブラリを再構築してください。

R4.50 に於けるその手順例を以下に示します。

```
A>ar30　x　pidlib.src　<CR>
A>cl30　-mb　pidinit.c　<CR>
A>asm30　pidcalc.asm　<CR>
A>asm30　pidcalc2.asm　<CR>
A>ar30　r　pidlib.lib　pidinit.obj　<CR>
A>ar30　r　pidlib.lib　pidcalc.obj　<CR>
A>ar30　r　pidlib.lib　pidcalc2.obj　<CR>
```

付録　D：互換C関数ソース

本製品の関数、PIDcalc()，PIDcalc2() はアセンブラで記述されていますが、PIDLIB.SRC にこれと同等の演算を行なう、C言語で書かれたライブラリソースが含まれています。アセンブラ版と全く同じアルゴリズムで計算を行なっている為、演算手順の理解や、他のプラットホームへの移植や評価等に利用できます。

ライブラリの取りだしは、PIDLIB.SRC　をカレントディレクトリに置き、

A>ar30　x　pidlib.src　pidcalc0.c　<CR>

としてください。カレントディレクトリに PIDCALC0.C が作成され、その中に、上記２関数が記述されています。

・本ソフトウェア及び本書は、著作権法により保護されています。
無断で複製・改変することはできません。
・本ソフトウェア及び本書の内容は、製品改良等のため予告なしに
変更されることがありますので、ご了承ください。

ＰＩＤ制御ライブラリ
ＡＤＳＰ３２４－２９
取扱説明書

初版　　1993.12
第３版　1995.11

中部電機株式会社
〒440-0004 豊橋市忠興 3 丁目 2-8
TEL (0532) 61-9566
FAX (0532) 63-1081